Optimization of Multi-Stage Amplifiers in Deep-Submicron CMOS Using a Distributed/Parallel Genetic Algorithm

R. Santos-Tavares[1,2], N. Paulino[1,2], J. Higino[1], J. Goes[1,2], J. P. Oliveira[1,2]
Universidade Nova de Lisboa[1] / CRI-UNINOVA[2]
Campus FCT/UNL, 2829 – 517 Monte da Caparica – PORTUGAL
rmt@uninova.pt

***Abstract*— This paper presents a framework for time-domain optimization of amplifiers employing a parallel genetic algorithm based on a message passing interface. This methodology achieves a considerable reduction in the optimization time (up to 19 times faster than a serial implementation). Increasing the processing capacity allows searching within a larger design space using complex transistors models, yielding more accurate results. The optimization, based on transient simulations, is possible due to the integration of a genetic algorithm optimizer together with the open-source simulator NGSPICE source-code.**

I. INTRODUCTION

The increasing specification requirements of CMOS mixed signal circuits (such as, increasing ADC resolutions and conversion rates) ultimately results in the need for operational amplifiers (opamp) with larger DC gains and larger gain-bandwidth product (GBW). In order to achieve these high-level specifications, using deep sub-micron technologies and reduced voltage power supplies, it is necessary to employ multiple gain stages. This way is possible to overcome the reduced drain-source resistance, r_{ds}, values of short-channel MOS transistors. However, high order short-channel effects difficult the determination of the r_{ds} value as a function of the transistor drain-source voltage (V_{DS}). Therefore it is almost mandatory to use advanced device simulation tools and models, such as BSIM3v3 [1], in order to obtain acceptable results.

Cascading several gain stages implies the use of complex compensation techniques to obtain stable opamps with a large GBW value. The resulting opamp transfer function will have several poles (some of them complex conjugated pairs) and zeros, making the amplifier design a complex task. Therefore the final design accuracy depends on the availability and quality of a powerful optimization tool.

Previously, an optimization tool based on genetic algorithms was presented in [2]. In order to simplify the optimization procedure this tool uses a time domain approach instead of the more traditional frequency domain approach. For example, in switched-capacitor circuits the objective is to have a stable opamp with a given settling error, after a given available time. By analyzing the step response of the opamp it is possible to obtain a single key performance indicator (KPI) that encloses all the traditional indicators, such as DC gain, GBW and phase margin. Following this approach, the opamp design can be accepted just by checking if the settling error is smaller than the desired value and that the step response is stable.

The time-domain step-response, $h(t)$, can be obtained from the closed-loop transfer function of the amplifier, $H_{CL}(s)$, multiplying it by $1/s$ and applying the inverse Laplace transform. The numerical value of $h(t)$ is based on the DC operating point of the amplifier. However, in this approach, it is difficult to include the effect of the voltage variation on the r_{ds} value of the transistors. To have a more exact design procedure, it is necessary to perform a transient simulation, which can consume a lot of CPU time.

In this work, the transient simulation is obtained using the open-source NGSPICE simulator source-code [3], with BSIM3v3 models. As already stated, this simulation is very accurate since it takes into account non-linear effects in the MOS transistors, but it can be very time consuming. This is especially problematic for a genetic algorithm optimization where each member of the population must be evaluated, resulting in a very large number of transient simulations.

Parallel and distributed computing (PDC) systems are recognized as an efficient computer architecture that can potentially improve the global processing performance when compared to single-processor/single-process systems [4]. The objective of the work, described in this paper, is to obtain an accurate and fast optimization tool for high performance opamps designed in advanced sub-micron technologies. The tool uses an enhanced optimization kernel, presented in [5], adapted to a parallel and distributed computation environment. This genetic algorithm is centrally controlled in a master computer and the evaluation of each opamp from the population, is distributed to each available slave computer using a standard Message Passing Interface (MPI).

The process of controlling the genetic algorithm consists of generating the population of opamps, and sending each of these elements for fitness evaluation in the slaves. By collecting this information, the population is ordered (ranked) and a new population is produced for the next generation. These steps are relatively fast because they do not use much CPU. Since a chromosome with several thousand bits of length can describe each opamp in the population, the communication with the slaves is also a simple procedure. The more time consuming step is the time-domain response of the opamp that requires a transient simulation. This step can be performed independently in parallel using as many CPUs as possible to reduce the overall optimization time. The distributed/parallel kernel runs on a network of desktop computers taking advantage of the time periods when these computers have a free time slot available. This way, it becomes possible to achieve high computing capacity without acquiring new/specific costly hardware.

978-1-4244-1684-4/08/$25.00 ©2008 IEEE

This paper is organized as follows. Section II describes our parallel/distributed genetic algorithm architecture, and in section III is presented the distributed/parallel approach implementation. In section IV are shown the optimization parallel/distributed kernel results, and the optimum circuit simulated results. Finally, section V draws the main conclusions.

II. DISTRIBUTED/PARALLEL GENETIC ALGORITHM

The concept of the distributed genetic algorithm used in this work is based in a simple topology where one master central computer controls the genetic algorithm execution in several remote slave computers, as shown in Fig. 1. It starts by randomly generating a new population of opamps, and each individual opamp circuit is encoded into a chromosome. It then sends a set of chromosome to the slave computers, where a transient simulation of the opamp is performed and the circuit settling time is measured. This information, together with the fitness result of the opamp, is returned back to the master computer. By collecting this information, the population is ordered (ranked) and a new population is created for the next generation. To create a new population, the master uses mutation and cross-over operators randomly applied to selected elements of the current population. The random selection is skewed so that the elements with the highest fitness have a higher probability of being selected.

After finishing the set of circuit simulation (one or more), the slave computer receives, from the master computer, a new set of circuits (one or more) to process. This procedure is repeated until the fitness of all the elements in the population is computed. After a certain number of generations the procedure is stopped and the netlist of the element with the best fitness in the final population is generated, corresponding to the optimization output.

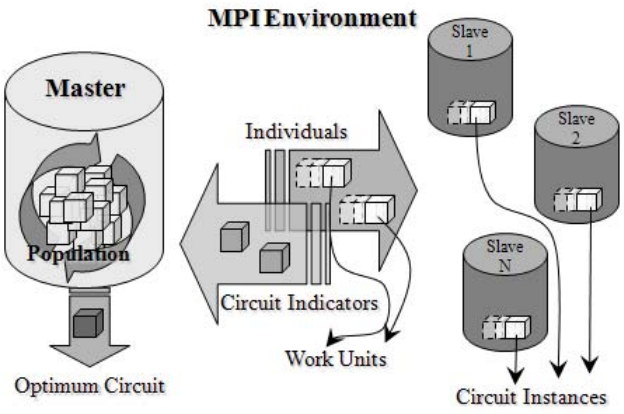

Figure 1. MPI Implementation of the distributed/parallel system

The information needed to describe each opamp circuit is encoded using the chromosome structure shown in Fig. 2. This chromosome is constituted by the following genes: the drain current (I_d), channel width (W) and length (L) for each transistor, three opamp compensation capacitors, C_a, C_b, C_m and two satellite compensation capacitors, C_{sN}, C_{sP}. In the selected example the chromosome is 992 bits long.

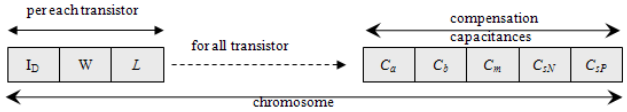

Figure 2. Chromosome description.

The fitness of each opamp is calculated using (1), which compares the actual performance of the circuit with the desired performance. It takes into account the settling-time (ST), the total current consumption (I_{TOTAL}), and total compensation capacitance (Cc_{TOTAL}) for a given opamp. All these three performance parameters have to be minimized.

$$f(ST, I_{TOTAL}, Cc_{TOTAL}) = (1 - e^{-\frac{ST\_desired}{ST\_achieved}})(1 - e^{-\frac{I_{TOTAL}\_desired}{I_{TOTAL}\_achieved}})(1 - e^{-\frac{Cc_{TOTAL}\_desired}{Cc_{TOTAL}\_achieved}}) \quad (1)$$

When compared to the single-processor/single-process approach described in [2], the new architecture proposed in Fig. 1, achieves several new advantages summarized by:

- The transient simulations are executed independently in parallel and in separate computers.
- In order to speed-up the optimization time more computers can be added. These can have different hardware configurations and processing capabilities.
- It allows to use computers that are not 100% dedicated to the optimization engine, but still able to help the job, e.g., desktop computers.
- The hardware costs of a single multiprocessor machine, capable of running the optimization procedure in the same time are much higher than using this approach.
- Due to the reduction of the optimization time, a larger population can be used in the genetic algorithm, thus increasing the search capability of the design space by the genetic algorithm and therefore improving the optimization results.

III. MPI IMPLEMENTATION

The implementation of the mater-slave topology presented earlier, in Fig. 1 is based on a Message Passing Interface (MPI) platform [6]. Using MPI it is possible to build a simple time-domain optimization distributed environment, capable of launching and controlling multiple processes of intensive computation.

MPI framework handles work load distribution, according to the slave computers performance. Initially, it distributes a preconfigured set of opamps, *work unit* (one or more individuals), to all slave computers, as shown in Fig.1. Then, if there is a slower machine, the faster machines will process the remaining *work units*. The evaluation finishes when the slowest slave machine concludes its last individual evaluation and sends its results to the master.

A. The Master Process

As explained before, the master process, depicted in Fig. 3 manages the genetic algorithm itself: it creates, distributes

and receives all the data to and from the slave machines. Based on the MPI configuration, it sets up *N* slave machines and sends them *work units* to be processed.

The first step is the generation of the population of circuits, according to the chromosome configuration. Then the following steps are repeated in each generation:

1. Distribution of one or more chromosomes through the slave machines as packages, *work units*, which are mapped to circuit netlists for transient simulation;
2. Evaluation of circuits, in the slaves;
3. The master computer gathers the *circuit results*: the fitness and the circuit simulated measures;
4. Apply selection, crossover and mutation operators to generate a new population;

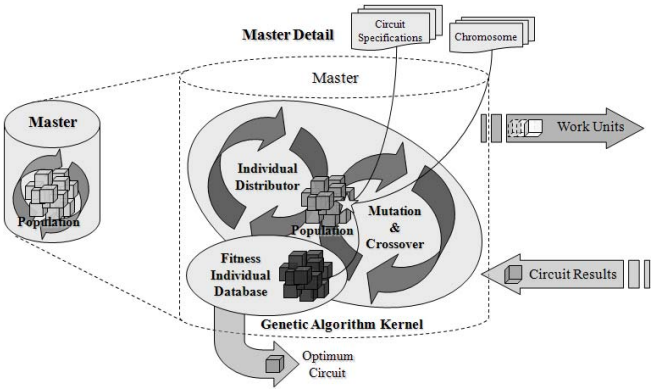

Figure 3. Master process from MPI Implementation of the distributed/parallel system

The previous cycle continues through a given number of generations. At the end, the netlist of the element of the last population with the best fitness is outputted.

B. The Slave Computer(s) Process(es)

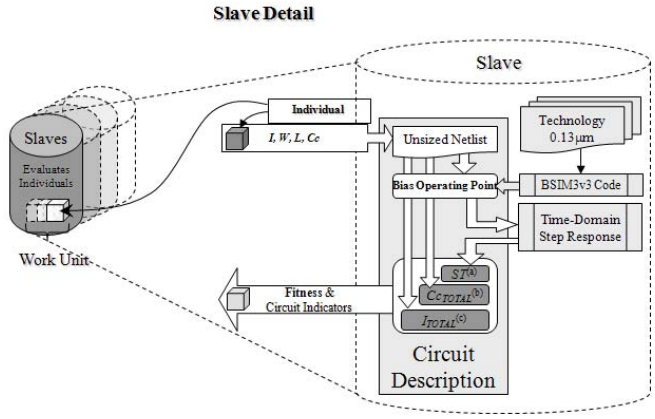

Figure 4. Slave processes from MPI implementation of the distributed/parallel system

Each slave computer executes the following steps:

Step 1. Set the size of the devices, and all other circuit variables (currents and compensation capacitances) with the values from the chromosome;

Step 2. The transient simulation is executed;

Step 3. The circuit fitness function is calculated using (1);

Step 4. Send the fitness and circuit information to the master.

In this work, each slave process can only evaluate one individual at a time because a single thread application is implemented. Although, several processes can run in parallel on a multiprocessor slave machine. The process parallelization is handled by the MPI framework.

IV. Optimization and Simulation Results

A. Proposed Circuit

To validate the proposed methodology, the amplifier presented in Fig. 5 was designed and optimized for a 130 nm standard CMOS technology with $V_{tp} \approx$ -0.33 V and $V_{tn} \approx$ 0.38 V. The circuit was optimized to operate with a supply voltage value equal to 1.2 V and to be used in a front-end Sample-and-Hold (S&H) of a 14-bit 25MS/s Pipeline ADC (with a feedback factor $\beta \approx 1$ and normalized sampling loading capacitances of about 24 pF). The settling-time specification is less than 20ns for an accuracy better than 0.004% (corresponding to an error smaller than 20µV assuming a differential reference voltage of 500mV). For higher or lower load capacitances, the optimized amplifier can be scaled linearly (*W's* and I_D*'s*), up and down, respectively.

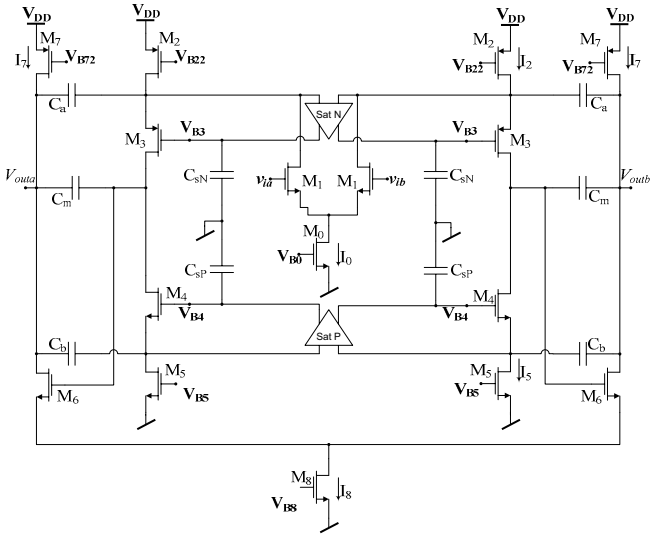

Figure 5. Low-voltage two-stage cascode-compensated amplifer.

TABLE I. Simulation Results

	Results
Open-Loop Gain, A_{OL}	>106 dB
Power Consumption, P_T	9.64 mW
Capacitance Area, C_{AREA}	1.88 pF
Settling-time, ST	15.7 nSec
Output Swing, OS	913.5 mV

TABLE I. resumes the NGSPICE [3] electrical simulation results of the opamp with the optimum transistors sizes (*W*, *L*) and compensation capacitor values, obtained after running the

optimization engine for 1000 generations and with a population size of 100.

Fig. 6 plots the output differential response of the amplifier. The differential input is a step with 500mV, centered at 800mV (input common-mode voltage). After a few clock steps, the differential output reaches the amplitude of, approximately, 500mV. Fig. 6 depicts the point where the signal enters the error margin range (higher than 499.98mV and less than 500.02mV) in approximately 15 ns (at time instant 2.650µs).

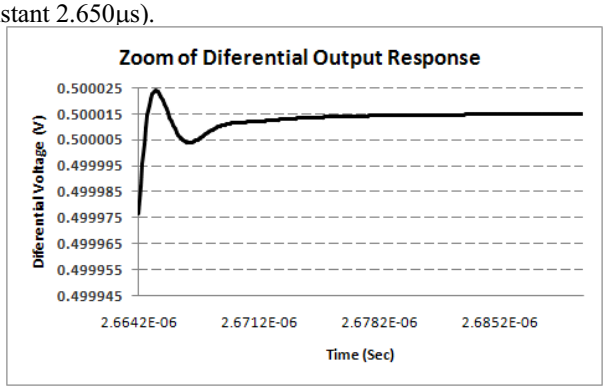

Figure 6. Simulated settling-response of the optimum opamp

B. *Distributed/Parallel Environment Performance*

The algorithm performance tests were conducted using a cluster of computers with different hardware configurations: five Pentium 4@1.7GHz, with one CPU logical core, named: pvm6, pvm7, pvm8, pvm9, pvm10; four Pentium 4@3.0GHz, with two CPU logical cores, named: pvm1, pvm2, pvm3, pvm4 and one AMD Semptron @ 2.8GHz (the master), with one CPU logical core, named pvm5.

To assess the performance of the parallel implementation versus the serial version of the circuit optimizer, a *speedup* factor is defined as:

$$Speedup = \frac{T_{Serial}}{T_{Parallel}} \quad (2)$$

where T_{Serial} is the time necessary to execute the optimization on a single machine (P4@3.0GHz) with a serial version of the genetic algorithm. The $T_{Parallel}$ is the time necessary to execute the optimization by the distributed/parallel version.

TABLE II. SPEEDUP FACTOR VS. NUMBER OF GENERATIONS VS. NUMBER OF INDIVIDUALS

Population Size \ Number of Generations	100	300	500	800	1000
50	13.97	13.50	13.94	14.02	13.93
100	14.42	14.39	14.37	14.30	14.46
500	18.76	18.73	18.37	18.27	18.47
1000	19.27	19.19	18.82	18.73	18.93

TABLE II. shows the speedup factor for different combinations of the number of generations and population sizes. The results show that the speedup factor does not change appreciably with the number of generations and it increases with the population size. This is expected because in each generation the individuals of the population are evaluated in parallel and the population of each generation is evaluated after the previous (in a serial way). These examples were obtained using all the machines (10 computers) in the cluster.

The *speedup* as function of the number of slave computers is shown next in TABLE III. In this test the following computers were used: 1 slave pvm5; 2 slaves pvm5, pvm1; 4 slaves pvm5, pvm1, pvm2, pvm3. The 10 slaves test was conducted with all machines. These tests were executed with a population size of 100 individuals and for 100 generations.

TABLE III. SPEEDUP VS. NUMBER OF COMPUTERS (SLAVES)

Number of Computers (slaves)			
1	2	4	10
1	3.90	7.46	14.42

The previous tests show that the parallel implementation of the genetic algorithm is much faster than the serial implementation. Depending on the population size it can be up to 19 faster if 10 slave computers are used. The speedup factor scales almost linearly with the increase of the number of slave computers.

V. CONCLUSIONS

This paper presented a framework for time-domain optimization of amplifiers employing a parallel genetic algorithm based on MPI. This methodology achieves a considerable reduction in the optimization time. Increasing the processing capacity allows searching within a larger design space using complex transistors models, e.g. BSIM3v3 yielding more accurate results. The optimization, based on transient simulations, was possible due to the integration of a genetic algorithm optimizer together with the open-source simulator NGSPICE source-code. This framework also permits to reuse hardware such as desktop computers.

ACKNOWLEDGMENT

The research work that led to this implementation was partially supported by the Portuguese Foundation for Science and Technology (FCT/MCT) under GRANT SFRH/BD/22798/2005 and under projects LEADER (PTDC/EEA-ELC/69791/2006), SIPHASE (POSC/EEA-ESSE/61863/2004) and SPEED (PTDC/EEA-ELC/66857/2006).

REFERENCES

[1] BSIM3v3.2.4 MOSFET Model - Users' Manual, UC Berkeley - 1999; http://www-device.eecs.berkeley.edu/~bsim3/

[2] R. Santos-Tavares, et. Al., "Optimum Sizing and Compensation of Two-Stage CMOS Amplifiers Based On a Time-Domain Approach", IEEE ICECS'2006, France, 2006.

[3] NGSPICE: a mixed-level/mixed-signal circuit simulator. http://NGSPICE.sourceforge.net

[4] G. C. Fox, R. D. Williams, and P. C. Messina, "Parallel Computing Works!", Morgan Kaufmann Publishers, 1994.

[5] B. Vaz, R. Costa, N. Paulino, J. Goes, R. Tavares, A. Steiger-Garção, "A general-purpose kernel based on genetic algorithms for optimization of complex analog circuits", MWSCAS 2001, 2001.

[6] Open MPI: A High-Performance, Heterogeneous MPI - 1-4244-0328-6/06 2006 IEEE; http://www.open-mpi.org/papers/heteropar-2006/heteropar-2006-paper.pdf